A New View on What Limits TCP/IP Throughput in Local Area Networks

Marc Abrams
Qizhong Chen

TR 91-23

Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24061

December 31, 1991

A New View on What Limits TCP/IP Throughput in Local Area Networks

TR 91-23

Marc Abrams
Qizhong Chen
Computer Science Department
Virginia Polytechnic Institute and State University
Blacksburg, VA 24061-0106
U.S.A.
abrams@cs.vt.edu

24 December 1991

Abstract

This paper presents experimental results on what limits local area network throughput at the application program level for two popular transport protocols, TCP and UDP, using two application program interfaces, Berkeley sockets and System V transport layer interface. The sensitivity of application-level performance to the choice of host computer speed and background load on the host are also studied. Two sets of measurements are discussed. The first contains macroscopic measurement of throughput on 68020, 68030, 68040, 80386, SPARC, and MIPS R2000 and R3000 based computers over a single Ethernet subnet. The second presents a detailed timing analysis using a hardware monitor of a TCP/IP implementation for PC architecture computers. Previous studies implicate memory copying, checksumming, and the operating system interface as the major overheads in TCP/IP, rather than the time required to execute the protocol itself. This study indicates that these factors are secondary when the sender and receiver are closely matched in speed; rather the primary bottleneck is the TCP flow control mechanism. TCP flow control becomes closer to optimal as the degree of speed mismatch increases. We draw conclusions on why window mechanisms should be augmented by rate based flow control in the new generation of high data rate networks.

1 Introduction

This paper presents the results of a detailed measurement study of the TCP/IP protocols operating on a variety of architecture computers over an IEEE 802.3 LAN. The results confirm through measurements past results in the literature, and brings to light a new view on what limits TCP/IP throughput.

The study consists of macroscopic followed by microscopic measurements. The macroscopic study (Section 2) investigates application-level performance sensitivity to three factors: host speed, host load, and choice of application program interface (Berkeley sockets versus System V transport layer interface (TLI)). The microscopic study (Section 3) then analyzes what limits the performance of one particular TCP/IP implementation. The implementation chosen is a relatively slow host (an Intel 80386-based PC), because the degree to which a slow ($\approx$1 MIP) host can be pushed to utilize a 10 Mbps (10^6 bits per second) LAN indicates how a host that is 10 to 100 times faster can utilize a LAN whose data rate is 10 to 100 times faster.

1.1 Related Work

Several works reporting the performance of the TCP(UDP)/IP protocols are discussed below.

Based on the observations of distributed graphics applications running on the V distributed operating system, Lantz shows the effect of the following parameters (listed in order of their importance) on the performance of data transmission: speed of the source machine, speed of the destination machine, choice and implementation of network transport protocol (e.g., choose a general-purpose transport protocol versus a specialized protocol; implement the transport process within the operating system kernel versus outside the kernel), and pattern of data transmission (i.e. the granularity of each data transfer).[8]

Svobodova [14] reports the measured throughput and delay of several transport layer proto-

cols. These measurements indicate that performance improvements (throughput and delay) of various OSI TP4/CLNS and TCP/IP implementations can be achieved by implementation optimizations. A major TCP/IP performance improvement is achieved by a combination of many changes:

- elimination of data copies into internal TCP buffers,
- modification of network drivers to allow more than one packet to be queued for transmission,
- prediction of the header of the next packet to be sent or received, and
- caching of frequently used data structures.

The author concludes that good design of communication software, clever coding of the protocol, and the support of the operating system play a critical role in improving the transport service.

Cabrera, Hunter, Karels, and Mosher study user-perceived performance when using TCP and UDP on 4.2 BSD Unix in an Ethernet environment.[2] They assess the impact that different processors, network hardware interfaces, Ethernets, and host load have on the performance. The host load and network hardware interface have a severe effect on the user processes' perception of network throughput. They also analyze the 4.2 BSD Unix TCP and UDP implementations and find that data copying and checksumming account for a disproportionate share of the total delay time.

This paper also assesses how different processors and host loads affect the performance of current workstations and protocol implementations in a variety of operating systems (4.3BSD, SVR4, Mach 2.1, MS-DOS, and SunOS 4.1). We also measure the effect that varying application program interfaces have on the performance.

Clark, Lambert, and Zhang make two conclusions on the TCP window-based flow control strategy.[4] First, the flow control mechanism is vulnerable to transmission errors and delays because the window mechanism combines both data flow control and error recovery. When the transmission error rate and the network delay is high, the sender must stop frequently to wait for an acknowledgment. Under such a situation, the window mechanism no longer controls the flow. Second, the window mechanism uses the receiver advised window size as the control parameter. But the advised window size does not carry enough information for flow control. This is because the window controls how much data can be sent rather than how fast the transmission should go. Transmissions occurring at an unregulated rate can easily congest the network and the receiver. In this case the window mechanism negatively influences performance.

This paper complements Clark *et al.* by exposing flow control and receiver advised window size problems in a low error rate, low delay environment, namely a LAN.

Sanghi, *et al.* describe an instrumentation of 4.3BSD Unix TCP/IP.[11, 12, 13] They conclude that a high resolution clock is essential to obtain good round trip time estimates, and that the round trip time estimator suggested by Jacobson [6] performs better than the one suggested in the original TCP specification.[9]

Clark describes implementation strategies to avoid the so-called silly window syndrome (SWS) and excess acknowledgements.[3]

Tips on efficiently implementing TCP/IP on personal computers are given by Saltzer.[10] The upcall is proposed to avoid excessive interface code from layering. Enhancements of DOS necessary to support network applications are also discussed.

Clark, Jacobson, Romkey, and Salwen investigate whether the transport layer is the bottleneck of data transmission.[5] The technique used is to identify the normal path through 4.3BSD Unix TCP and count the instructions. Based on their study, they predict that TCP can be used

on high speed fiber optical networks if implemented properly. The major overhead of TCP/IP packet processing comes from the operating system interface, which handles interrupts, restarts I/O devices, wakes up processes, and sets timers. Another overhead is operations that touch bytes, such as memory copy and checksum.

The microscopic measurements of Section 3 complement Clark *et al.* Section 3 instruments and measures one implementation to analyze sources of overhead in segment processing. Our study concurs with the 4.3 study conclusion that the TCP/IP protocol itself is not a bottleneck. Our measurements concur with the overheads of memory copying and checksumming. However we also analyze to what degree the window flow control mechanism limits performance, and find that the primary bottleneck arises from the window flow control mechanism, which forces idle periods in network hosts if the TCP receive buffer is improperly sized.

2 Macroscopic Measurements

The objective of the experiments reported in this section is to study the delay and throughput experienced by a bulk data transfer application at the application level as a function of host processor speed, host load, and choice of application programming interface.

2.1 Experimental Environment

The experiment is to transfer a 40K byte file using a data sending program on one host and a data receiving program on another host. The sending program transfers a file using a variety of write sizes in a tight loop; the receiving program reads the file in a tight loop. The write size is defined to be the number of bytes passed as an argument from the application program to the *write* C library function call.

A file size of 40K is chosen because the performance measures, described shortly, do not vary much if the file size is increased, except as noted in Section 2.3; however shorter file lengths

do affect the performance measures. Write sizes of 10, 100, 512, 1024, 1200, 1460, and 2048 bytes are used. Additional write sizes are used as necessary to determine the shape of measured throughput curves. Write sizes smaller than 10 bytes are not used to avoid excessive numbers of datagrams to transmit the file, which would unreasonably increase the duration of experiments. Write sizes of 512, 1024, 1460, and 2048 are used because 512 and 1024 are common disk blocking sizes, 1024 and 2048 are often factors of buffer sizes in many TCP implementations, 1460 is the maximum segment size that fits in a single 802.3 network packet, and 2048 is a size larger than the maximum segment size. Write sizes of 100 and 1200 are also used to better understand the shape of measured curves.

Each data point reported in this section represents the mean and confidence interval for a 95% confidence level. Each data point is based on 100 executions of the file transfer. This value is sufficiently large to keep the relative precision under 15% for almost all data points, yet the value does not make the duration of experiments unreasonably long.

The performance measure used is *throughput*, which is defined as the length of the transmitted file (40K bytes) divided by the delay. *Delay* is defined as the interval from when the receiving host receives the first packet of the transmitted file until the host receives the final packet. Delay excludes the time required for connection establishment and release.

Table 1 lists the machines used in the experiments, along with mnemonics identifying each machine. This set of machines represents a wide range of ratios of host to network speeds and encompasses different generations of architectures, including CICS microprocessors (Motorola 68020 and 68020, Intel 80386), RISC processors (SPARC, MIPS R2000 and R3000), and one CICS architecture benefiting from RISC design techniques (Motorola 68040).

In all experiments two hosts are connected to the same IEEE 802.3 10 Mbps CSMA/CD subnet. Experiments are perturbed by two factors. The first is system daemons that periodically

Mnemonic	*Machine Type*	*CPU*	*Operating System*	*TCP Receive Buffer Size*
D310	Dell 310	Intel 80386, 12.5 MHz	MS-DOS 4.0	0-4 Kbyte
PS2	PS/2 model 80	Intel 80386, 12.5 MHz	MS-DOS 4.0	0-4 Kbyte
MacII	Macintosh II	Motorola 68020, 16 MHz	A/UX 2.0 (SVR2 based)	4 Kbyte
A3000	Amiga 3000UX	Motorola 68030, 25 MHz	AT&T SVR4	4 Kbyte
NeXTcb	NeXTcube	Motorola 68040, 25 MHz	NeXT Mach 2.1	4 Kbyte
NeXTst	NeXTstation	Motorola 68040, 25 MHz	NeXT Mach 2.1	4 Kbyte
SUN	Sun 4/390	SPARC CY7C601	SunOS R4.1	4 Kbyte
DEC2100	Decstation 2100	MIPS R2000, 25 MHz	Ultrix 4.1	N/A
DEC5000	Decstation 5000	MIPS R3000, 25 MHz	Ultrix 4.1	16 Kbyte

Table 1: Machines used in Section 2.

awaken on the hosts being measured. The second is that traffic on the Ethernet during the experiments is not controlled. To minimize the second factor, experiments are performed in early morning hours when no user processes are running on any machines attached to the subnet. To assess the representativeness of the measured data, one version of the testing program, TCP with sockets, was run between two A3000's on three consecutive days. Using a writing size of 1460 bytes, the testing program transmits 40K bytes of data 100 times. The highest average transfer delay obtained from the three day experiments is 10% higher than the slowest.

2.2 Application-level Performance Sensitivity to Host Speed

The objective in this subsection is to identify the sensitivity of application level throughput to host processor speed.

2.2.1 TCP Throughput

Figure 1 represents throughput for the file transfer application using a Berkeley socket interface with the TCP transport protocol as a function of write size for pairs of homogeneous hosts. The

PS2-D310 throughput for a 2K write size is excluded from the figure because the PS/2 crashes when *write* is called with a 2K byte argument.

In Figure 1, as expected, faster host processors yield higher throughput. Larger write sizes lead to higher throughputs, but only for write sizes below 1K. The DECstation throughput flattens out at 8.4 Mbps with a 1024 byte write size; its performance appears to be limited by the 10Mbps physical media data rate.

Throughput is limited by the TCP receive buffer size and TCP segmentation.

TCP receive buffer size: Based on the size of the currently available receive buffer, the destination host returns an advised window size in every acknowledgement to the sending host. The receiver advised window size is an upper bound on the sender window size. Therefore a smaller receive buffer can cause the sender to stop more frequently, and this introduces delay that limits the data transfer rate. The receiver buffer size is 16K on the DEC, 2K on the D310, and 4K on the remaining hosts. The buffer size can be varied on the D310, and this is explored later in Section 3.5. Because the receiver advised window size is set to the available receive buffer, throughput is higher at write sizes that are factors of the receive buffer size, such as 1K and 2K.

TCP segmentation: TCP waits to transmit data until either it completely fills the current segment or until the segment flush timer expires. Therefore a large write size tends to increase throughput because fewer calls are made to *write* and each TCP segment is more likely to contain the maximum amount of data, and therefore fewer packets are sent.

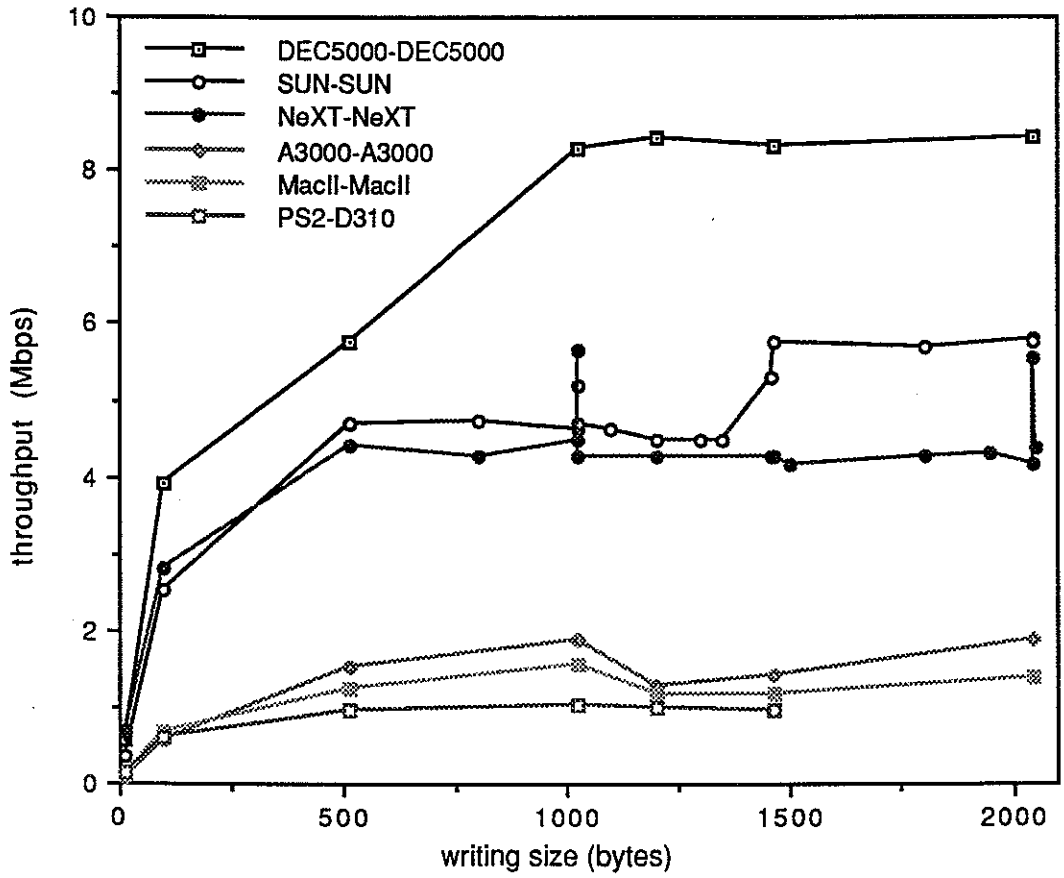

Figure 1: File transfer throughput using socket interface and TCP.

Write Size	PS2 to D310	MacII to MacII	A3000 to A3000	NeXTcb to NeXTst	SUN to SUN	DEC5000 to DEC5000
10	0.00%	0.01%	0.57%	0.05%	0.07%	3.70%
100	0.00%	0.00%	0.00%	0.61%	0.32%	0.03%
512	1.80%	0.01%	0.00%	0.05%	0.00%	0.09%
1024	20.70%	0.00%	0.00%	1.50%	0.00%	0.56%
1460	57.00%	0.10%	0.00%	0.17%	0.00%	0.07%

Table 2: Loss rates for UDP throughput experiments.

2.2.2 UDP Throughput

Figure 2 represents the throughput of the file transfer application using UDP, rather than TCP, as the transport protocol. Once again, the socket interface is used. Unlike TCP, the UDP throughput monotonically increases with write size. This data is easier to interpret than the TCP data for two reasons. First, there is no flow control, so that there is no artificial delay added to the communication due to waiting for acknowledgements to increase the upper edge of the send window. Second, there is no delay introduced by the transport protocol to fill segments; therefore the number of segments sent is inversely proportional to the write size.

With UDP, the host processor speed has a profound effect on throughput, because the throughput is only governed by the packet sending and receives of the source and destination hosts, respectively, and the physical transmission media bandwidth (10Mbps).

Table 2 shows the loss rates associated with each data point in Figure 2. The PS2 to D310 observations are excluded from Figure 2 because the loss rate at 1024 and 1460 bytes is 20% and 57%, respectively. In contrast the loss rates on other hosts never exceed 4%. This shows the disadvantage of using UDP when the sender is faster than the receiver.

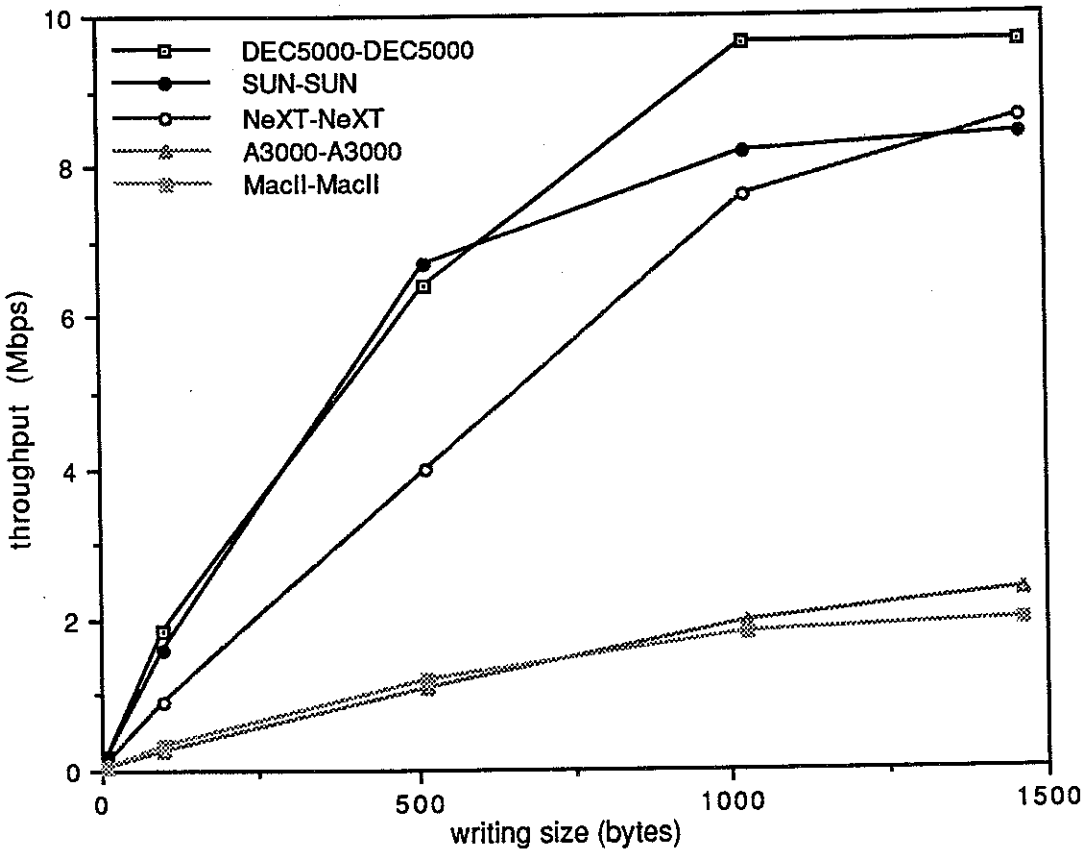

Figure 2: File transfer throughput using UDP and socket interface.

2.2.3 Comparison of TCP, UDP

The throughput of TCP and UDP is compared in Figure 3. (The MacII and A3000 hosts are excluded from the figure, but yield results similar to the other hosts.) At *small* write sizes, TCP is better because TCP delays transmission to fill segments, hence reducing the number of packets sent. At *larger* write sizes, using a reliable transmission medium with a sender that is no faster than a receiver, UDP wins, because the TCP window mechanism artificially introduces delay.

2.3 Application-level Performance Sensitivity to Host Load

A study of the sensitivity of the protocol performance to host load can confirm the contention in Clark *et al.* that the time required to execute instructions to process datagrams and segments by IP and TCP is not a bottleneck.[5] This is because transport-level throughput is insensitive to host load if and only if execution of the IP and TCP protocol software itself is not a bottleneck.

The experiment performed is to run a compute-bound computation on the sending host while the file transfer application program executes. The computation used is a matrix multiplication program that performs no I/O during the file transfer. We run zero, one, or three copies of the multiplication program; *load* is defined as the number of multiplication programs running. Therefore results from Section 2.2 correspond to the zero load case. Tables 3 and 4 present the results for the slowest hosts with TCP and UDP as the transport protocol, respectively. Faster hosts are not shown, because host load negligibly increased delay at all write sizes.

With TCP, the 16 MHz based 68020 MacII case shows a doubling and quadrupling of delay with write sizes of 10 bytes with loads of one and three. For 100 bytes, there is a negligible change in delay with a load of one, but a tripling of delay with a load of three. At 512 bytes the delay increases by 20% for loads of one and three. For 1024 bytes and above, the delay increases by a negligible amount. The faster 25 MHz 68030 based A3000 case shows negligible increase

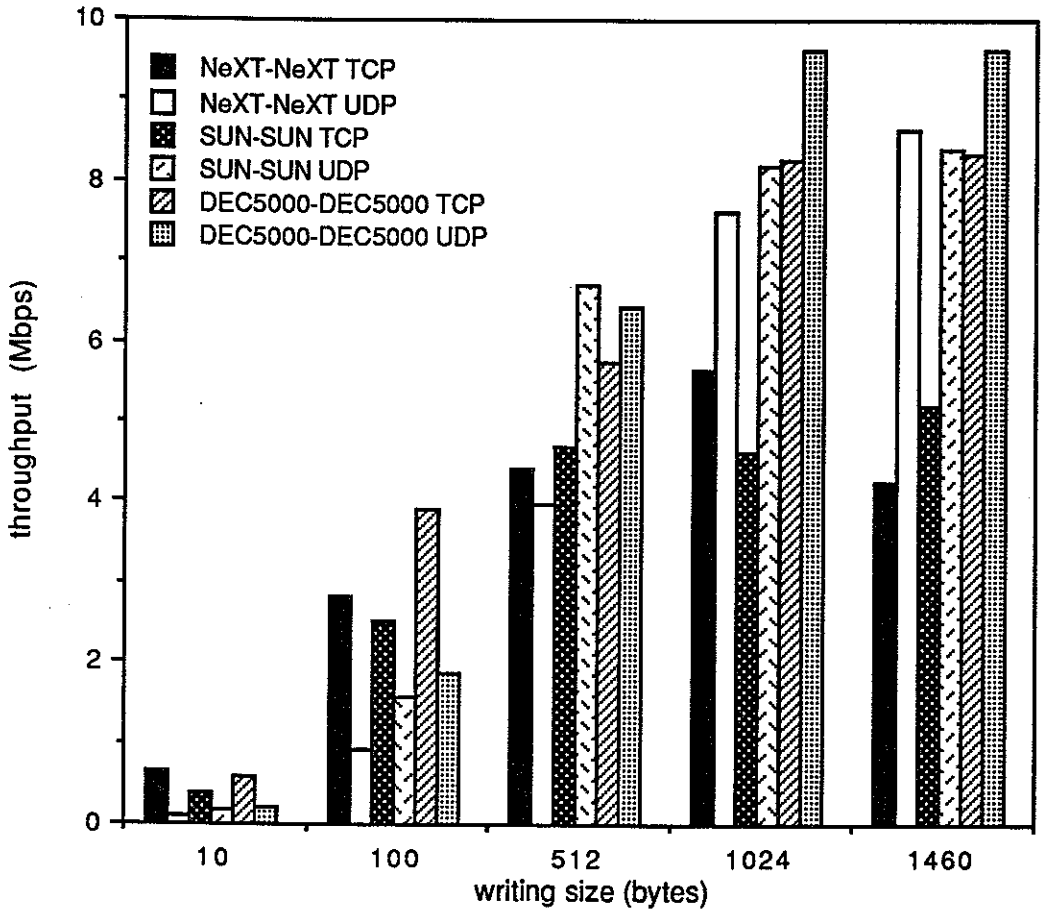

Figure 3: Comparison of TCP and UDP file transfer throughput using socket interface.

Write Size	*MacII to MacII*			*A3000 to A3000*		
	load=0	*load=1*	*load=3*	*load=0*	*load=1*	*load=3*
10	3015±12.6	5480±22.7	12200±91.8	4041±13.5	5072±103	12806±344
100	501±10.8	507±12.7	1521±37.9	594±6.3	597±6.6	675±39.4
512	266±8.3	319±12.9	328±14.3	216±1.0	230±14.6	236±16.7
1024	209±2.7	220±3.4	216±2.2	174±3.0	180±1.9	185±4.1
1460	292±12.2	324±17.6	332±13.3	254±14.8	265±15.1	279±15.3

Table 3: Data transfer delay, in milliseconds with a 95% confidence level, to send 40K bytes using a socket interface and TCP while various numbers of matrix multiplication programs are executing on the sending host.

Write Size	*MacII to MacII*			*A3000 to A3000*		
	load=0	*load=1*	*load=3*	*load=0*	*load=1*	*load=3*
10	8871±24.5	17585±32.3	38230±217	11920±14.3	21313±61.5	42301±1055
100	1021±7.2	1443±15.4	3684±34.7	1317±8.2	1279±4.8	1307±112
512	272±3.4	283±3.5	518±26.5	298±0.7	299±0.6	297±0.9
1024	181±3.1	197±3.1	210±7.5	166±0.6	168±0.7	167±0.6
1460	168±1.2	172±4.6	188±7.6	139±1.2	138±1.2	137±1.1

Table 4: Data transfer delay, in milliseconds with a 95% confidence level, to send 40K bytes using a socket interface and UDP while various numbers of matrix multiplication programs are executing on the sending host.

in delay as a function of load for all write sizes larger than 10 bytes. At 10 bytes, a load of one increases delay by 30% and a load of three triples the delay.

The UDP results follow a similar trend. For the MacII, a load of one or three and a write size larger than 100 bytes or 512 bytes, respectively, shows a negligible increase in the delay. For the A3000, delay only increased for a write size of 10 bytes.

Comparing UDP and TCP for the same hosts and write sizes, UDP delay is more sensitive to host load for small write sizes. This behavior is explained by considering the relative number of datagrams that results from each protocol. The number of datagrams sent by IP on behalf of UDP serves as an upper bound on the number of datagrams send on behalf of TCP, given a reliable transmission medium and equally matched sender and receiver speeds. At a write size of 10 bytes with UDP, IP sends 4096 datagrams for the 40K file transfer, while TCP generally results in less than 4096 datagrams because it delays transmission in an attempt to fill the current segment. Therefore UDP will be more sensitive than TCP to host load for a write of 10 bytes, as Tables 3 and 4 bear out. However with a write size of 512 bytes, UDP transmits only 80 packets, which is only 2% of the number of datagrams sent with a write size of 10 bytes, and both UDP and TCP are less sensitive to host load.

The conclusion from the measurements is that the total number of transmitted packets during a fixed time interval increases, the delay becomes increasingly sensitive to the host load no matter what write size is used. However the sensitivity decreases as speed of the host increases.

This conclusion implies that the results reported are sensitive to the size of the file transferred. For a write size that yields negligible delay increase in Tables 3 and 4, increasing the file size to the point where a thousand or more datagrams are sent will make delay sensitive to load. To verify this statement, the experiment reported in Table 5 is performed, consisting of a file transfer of 512,000 bytes using a write size of 512 bytes, which yields 1000 datagrams for UDP.

Protocol	*load=0*	*load=1*	*load=3*
TCP	3086±70.4	4806±93.3	8429±301.6
UDP	3533±60.0	6524±40.6	13775±196.9

Table 5: Effect of increasing size of file transferred on delay sensitivity to host load. All delays are in milliseconds, with a 95% confidence level.

In a 40K file transfer, the delay increases negligibly with load, but for the 512,000 byte file, the delay more than doubles with a load of three.

Based on these results, a file transfer that requires on the order of a thousand or more datagrams will be sensitive to host load, whereas file transfers requiring a hundred or fewer datagrams will be insensitive to host load.

2.4 Application-level Performance Sensitivity to Choice of Application Program Interface

The A3000 host implements both a socket and a System V TLI application program interface. Figure 4 compares the throughput of the file transfer application with both interfaces. Based on observation of only a single platform, we can make conclusions about the importance of choice of application program interface, but not about any inherent properties of the sockets and TLI that affect performance.

Figure 4 shows, as before, that TCP yields higher throughput than UDP at small write sizes, while UDP does better for large write sizes. However, TLI outperforms sockets when UDP is used, whereas sockets outperform TLI when TCP is used. The difference in performance can be surprising; for TCP and a write size of 100 bytes, the socket delay is 24% lower than the TLI delay, and for UDP and a write size of 512 bytes, the socket delay is 21% greater than TLI. Therefore a user must carefully evaluate alternative application program interfaces, ideally by performing experiments using the write sizes and timings of writes that the actually implementation will

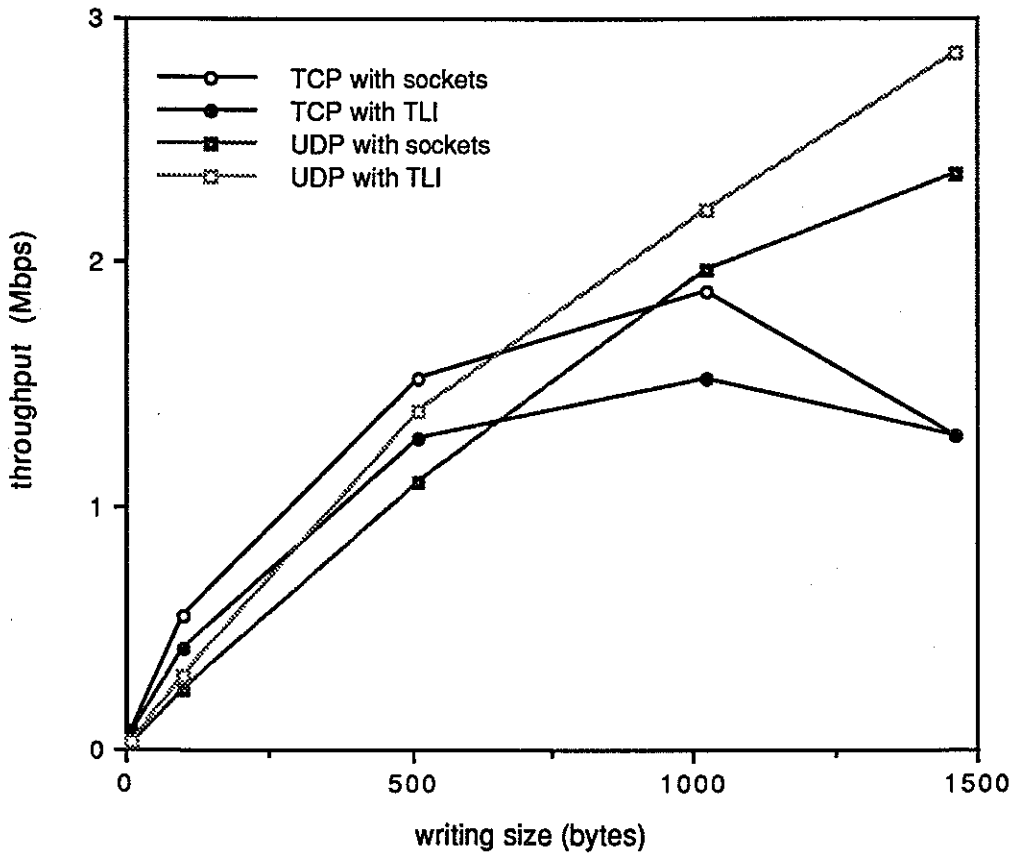

Figure 4: Comparison of socket and TLI interfaces on A3000 hosts.

generate.

3 Microscopic Measurements

Section 2 sets the stage for the detailed analysis of one particular protocol implementation in this section. Figure 1 shows that at one end of the performance spectrum, the DEC5000 can saturate an Ethernet. At the other end the 80386 based hosts never use more than 13% of the raw network bandwidth.

Does the 13% limitation arise from the host processor speed, or does it arise from something subtler, such as the protocol or its implementation? The answer is actually quite critical to the future of high speed networks. In terms of orders of magnitude, the question of whether two communicating 1 MIP, 1 Mbyte machines (e.g., a bit slower than the 12.5 MHz 80386-based D310) can fully utilize a 10Mbps network can be viewed as predictive of whether a pair of 10-100 MIP, 10-100 Mbyte RISC workstations can fully utilize a 0.1 to 1 gigabit per second network. How far can the slowest machine be pushed? The measurements discussed next suggest that even an 80386-based machine can be pushed to the limit of a 10Mbps network.

This section looks at detailed timings of a TCP/IP implementation distributed by IBM for IBM PC and PS/2 architectures, called MD-DOS/IP. The measurements are made on an Intel 80386-based machine: the D310 equipped with an Ungermann-Bass PC-NIC Ethernet adapter communicating with a 80386-based IBM PS/2 model 80 also running MS-DOS and MD-DOS/IP. The MD-DOS/IP package implements a Berkeley socket interface, and provides memory management, timer management, and multitasking extension to MS-DOS.

Even though RISC workstations generally run some version of the Unix operating system, while our measurements are from another operating system, we believe that they do have predictive value for Unix machines, because the multitasking system on MS-DOS interleaves execution

of protocol processes in a the same manner as Unix systems.

Our objective is to identify what limits throughput to about 1 Mbps in the D310 running MD-DOS/IP on a 10Mbps network.

3.1 MD-DOS/IP

The MD-DOS/IP package is implemented as a set of three tasks, for the application program, TCP, and IP, and an interrupt handler for incoming packets.

Outgoing packets: Each outgoing packet requires two copies, one from application to TCP task address space, and a second from TCP space to the network adapter. Outgoing packets pass from the application task to the TCP task, which executes outgoing operations of the TCP, IP, and adapter card driver functions, and then to the network adapter.

Incoming packets: Each incoming packet also requires two copies, first from the network to the interrupt handler, and then to the application task space. Incoming packets pass from the network adapter to the interrupt handler, then to the IP task, then to the TCP task, and finally to the application task.

3.2 Experimental Environment

Detailed measurements of the MD-DOS/IP package during execution of the file transfer application described in Section 2.1 are made. The MD-DOS/IP code running on the D310 is instrumented to log a sequence of timestamped events during the file transfer in memory space external to the application and protocol. Events recorded include the time at which each software module is entered and left and the time at which key protocol operations, such as setting or clearing timers and starting and ending the checksum computation, start and stop. Timestamps are provided by a 40-bit microsecond-resolution hardware clock card plugged into the D310 bus

that appears as a set of I/O ports [1]. Each clock read requires approximately 40 microseconds.

Any software measurement alters the measured data. To insure that the measurements accurately reflect the behavior of an uninstrumented system, several versions of MD-DOS/IP are compared: (1) the unaltered code, (2) the code with all instrumentation enabled, and (3) a set of codes with only instrumentation of certain modules enabled by compile time switches. The total delay for the file transfer for codes (1) and (3) differ by less than about 3% and are reported in the processing cost analysis of Section 3.3. The difference between (1) and (2) is larger; (2) is used only once in the paper, for the overal timing graphs given in Section 3.3.

3.3 Analysis of System-level Processing Costs

Figure 1 shows that the D310 achieves maximum throughput with a write size of 1024 bytes. Because our objective is to identify what limits the D310 throughput, we analyze the maximum throughput performance, and hence select a write size of 1024 bytes.

Figures 5 presents a portion of the time-dependent behavior of the sending side of the file transfer application and TCP/IP protocol for one particular observation with a write size of 1024 bytes. The diagram is representative of the entire log file of all observations of this write size. Discrete points on the y-axis of each graph represent specific activities of the sending and receiving host, respectively.

Sender behavior: Figure 5 partitions the activities of the sending host into six categories: application running, TCP task copying data from the application address space into a TCP buffer, TCP protocol running but not copying data from the application address space into a TCP buffer, IP protocol running, network adapter driver running, and host idle. Transmission of one eighth (5k) of the file is illustrated. As mentioned earlier, the application writes the data as 1K messages through the socket interface to TCP.

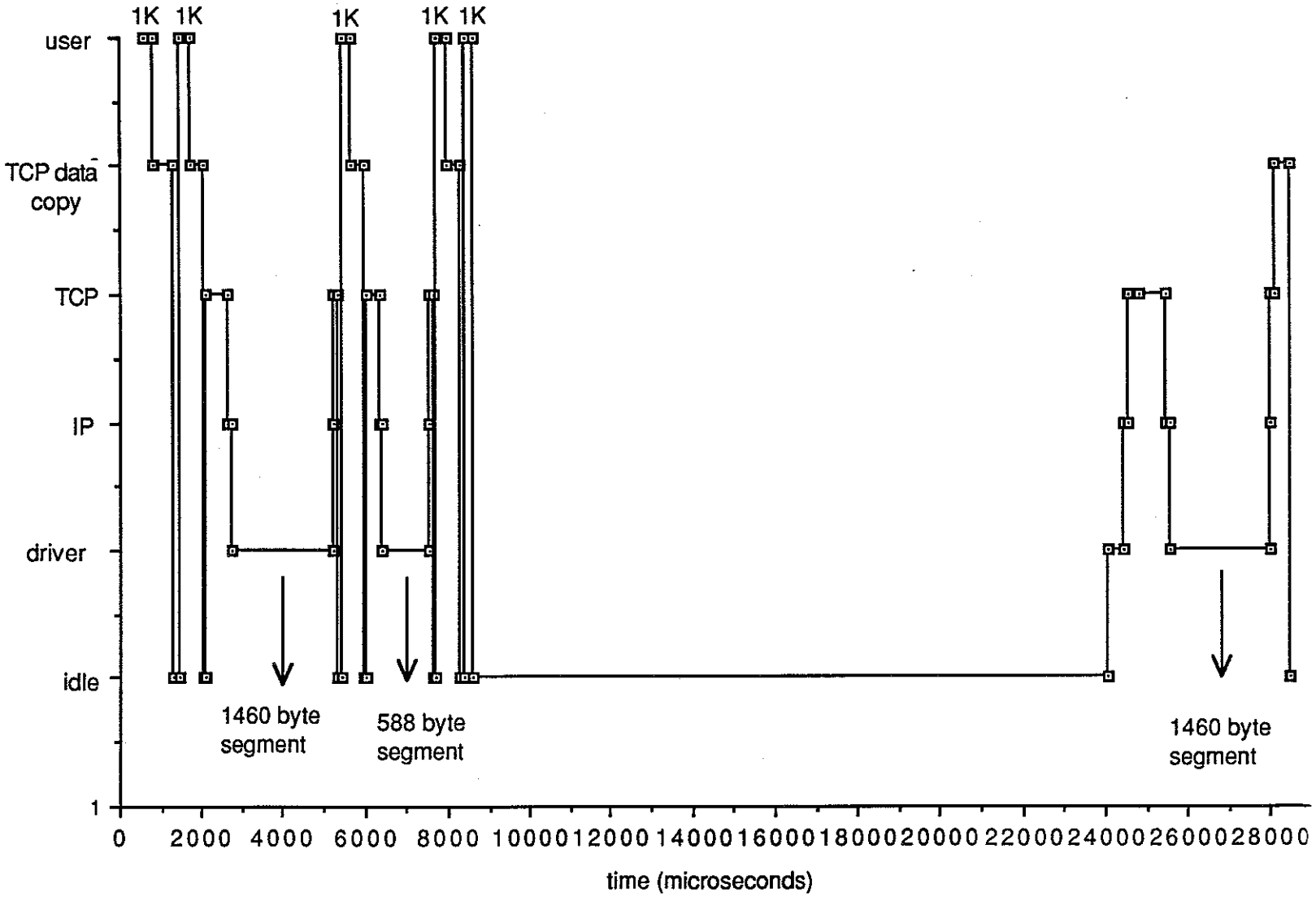

Figure 5: Time evolution of TCP/IP on the sending side for a transfer of 5K of data.

At time 600, the application writes the first 1K message. TCP copies the message to its address space, and sets a segment flush timer in an attempt to fill a 1460 byte segment, which is the maximum that will fit in one Ethernet packet. The host goes idle until the application task is rescheduled and sends the second 1K message. TCP copies the message to fill a 1460 byte segment, and 436 bytes of a second segment. The first (1460 byte) segment is passed to IP and then to the driver, which copies the resultant datagram to the network adapter.

At this point the TCP send buffer contains the last 588 bytes of the send application write. The size of the buffer between the application task and the sending TCP task as well as the TCP receive buffer size are 2K. Therefore the application task is scheduled and writes the third 1K message through to the TCP send buffer. Because the receive buffer size is 2K, and one segment of length 1460 has been sent but not acknowledged, the sending window is 588 bytes, and a second segment of size 588 bytes is passed to IP and through the adapter driver to the network. The send buffer now contains 1K of data, and the application task is scheduled to write its fourth 1K message to TCP. At this point the send window is closed, so that TCP cannot send more data. The only scheduleable task is now the application, which executes its fifth write of 1K to the socket interface. However the send buffer is now full, so that the application task blocks, and the host goes idle. The predominate feature of Figure 5 is that the host remains idle for 15.446 milliseconds until the driver receives an acknowledgement packet.

The MD-DOS/IP acknowledgement strategy is to delay acknowledgements when possible in an attempt to send one acknowledgement for every to received datagrams. Hence the sender receives a single acknowledgement increasing the upper send window edge by 2K. This permits TCP to create a third segment of maximum size, 1460 bytes, using the current contents of the TCP send buffer; the segment is transferred through IP and the driver to the network. With 1K of the TCP send buffer now free, the fifth application write, back at time 8607, can now be

copied to the TCP send buffer.

Analysis: Figure 5 indicates that the predominate component of delay is due to the lack of receive buffer space causing TCP to stop sending data. The secondary delay component is the device driver, which is copying data to the network adapter under CPU control. The time required for TCP, IP, copying data from the application to the TCP send buffer, and the application itself are all much smaller contributers to delay.

How significant is the idle time? The *idle* state is defined as the time when there is no scheduleable task. We instrumented MD-DOS/IP to timestamp only transitions between idle and non-idle states, and no other events. For thirty observations of the 40K byte file transfer application with a 1K write size, the sample mean of delay is 345.452±16.00 milliseconds, and the sending host is idle for 156.491±8.65 milliseconds, using a 95% confidence interval. (The sample mean of delay for a code without instrumentation is 10.4 milliseconds lower.) This yields a remarkably high 45.3% idle time, primarily due to the sender artificially blocking because no receiver buffer space is available.

This data suggests that the file transfer throughput will increase markedly if either the receiver has a larger buffer or if the window mechanism is changed to another flow control mechanism. These changes are investigated in Section 3.5.

Upper bound on throughput: The issue of which flow control mechanism maximizes performance is a subject of longstanding debate [7]. We use our instrumented code to compute an upper bound on the throughput that can be achieved by any flow control mechanism.

Suppose that all costs associated with file transfer are zero on the D310 except for the cost of executing the TCP and the IP protocols themselves. In terms of Figure 5, the user, TCP data copy, and driver times could at best be zero. An upper bound can be computed by multiplying

Protocol	*Sender*		*Receiver*	
	Send Data	*Processing ACK*	*Receiving Data*	*Sending ACK*
TCP	523±0.61	297±0.67	493±5.20	153±0.39
IP	124±0.53	112±0.46	106±0.32	118±1.25
Total	647	409	599	271

Table 6: Sample mean and confidence intervals of time required for TCP and IP protocols to process one 1K byte segment with a 95% confidence level. All times are in units of microseconds.

the measured cost per data and acknowledgement segment of TCP and IP at the sender and receiver times the number of packets of each type that are required for a 40K file transfer.

The measurements from Table 6 will be used to calculate an upper bound. Each entry in the table is based on observation of 100 repetitions of the following experiment: sender opens a connection to the receiver, then transfers 1K bytes using 1K bytes as the write size, and then closes the connection. The sender (respectively, receiver) measurements in the table are obtained by instrumenting the sending (receiver) code and running it on the D310.

Some assumptions are required for the upper bound. First we assume that TCP sends one acknowledgement segment for every two data segments received; this is the normally case for the file transfer application using MD-DOS/IP. Second we assume that TCP sends forty 1K segments to transfer the 40K byte file. Therefore it takes 851.5 microseconds ($647 + 0.5 \times 409$) at the sender and 734.5 microseconds ($599 + 0.5 \times 271$) at the receiver to transfer 1K of data. The sending path is the bottleneck, which leads to a data rate of 9.6 Mbps, exceeding the Ethernet data rate available after the bandwidth required for segment and datagram headers is removed.

3.4 TCP Segment Processing Overhead

Figures 6 and 7 decompose the cost of TCP segment processing. The measurements are made using the D310 as the sender (respectively, receiver) for Figure 6 (Figure 7). The experiment

performs 100 repetitions of the following: sender opens a TCP connection, sender writes x bytes of data using a single *write* call, sender closes connection. The values of x, the data size, used are 1, 10, 100, 500, 1024, and 1460 bytes. The connection is closed after each write to insure that TCP sends one segment for each call to *write*. The TCP implementation of MD-DOS/IP is instrumented to record the time spent performing data copy, checksum, timer management, and remaining TCP functions. (The upper bound computed earlier assumes that the first two times are zero.)

Timer management includes setting timers and processing acknowledgements. The sending side TCP starts a timer each time a segment is sent. On receiving an acknowledgement, TCP calculates the round trip time. Based on the new round trip time, TCP updates the retransmission time out value and sets the sending window size. These two values are set using Jacobson's round trip timing and slow start algorithms [6].

TCP send overhead: Figure 6 shows that the data copy and checksum costs grow linearly with the data size, which is the expected result. The cost of timer management is nearly constant, with a mean of 84 microseconds per segment, consisting of 22 microseconds for setting a timer upon segment transmission and the balance for acknowledgement processing. The cost of the remaining TCP operations is also nearly constant; its mean is 351 microseconds per segment.

At worst (a write size of 1460 bytes), data copying and checksumming each take about a third of the time (36% and 32%, respectively), with the remaining TCP functions consuming the remaining third of the time (31%). This result bears out suggestions in the literature of how to reduce copying and checksumming costs, such as by combining both functions into one loop [5]; eliminating the checksum [2]; and avoiding one data copy, from application to protocol address space.

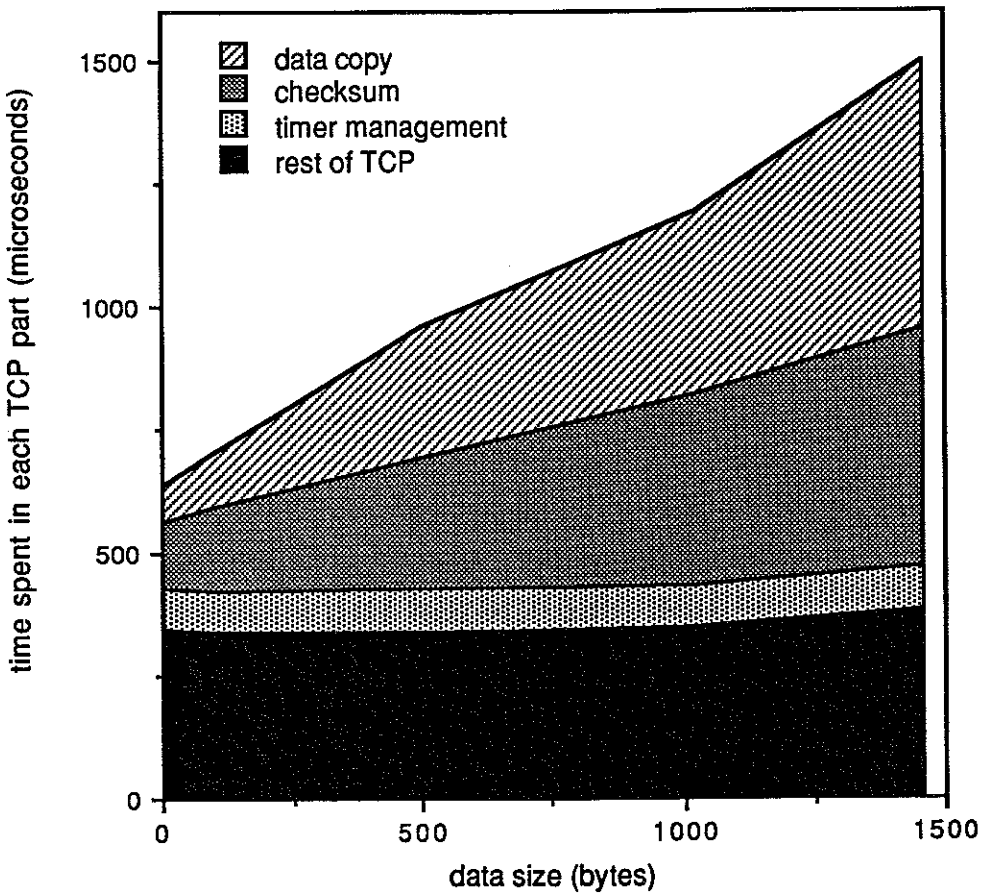

Figure 6: Overhead of TCP segment processing on the sending side.

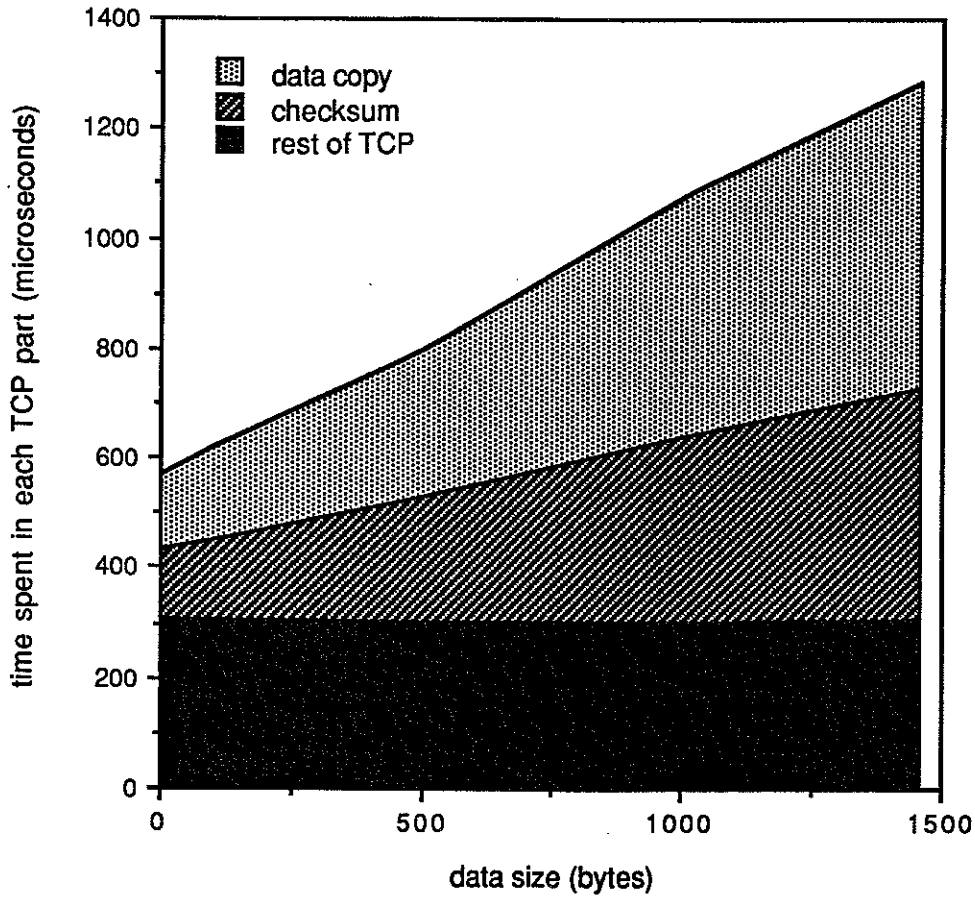

Figure 7: Overhead of TCP segment processing on the receiving side.

TCP receive overhead: Figure 7 partitions the receive cost into data copy, checksum, and the remainder of TCP. There is no timer management in the receiving side. Again, the first two costs increase linearly with the data size as expected. The remaining TCP costs are constant at just over 300 microseconds per packet. The suggestions listed earlier for the sending side can also be used on the receiving side to reduce the copying and checksumming costs.

3.5 Bottleneck Analysis

The preceding results suggest that the flow control mechanism, rather than the time required to execute the protocol, is the primary bottleneck in bulk data transfer applications with TCP/IP. This section considers the question of what limit throughput from a system level viewpoint, and expands the study to consider the sensitivity of the limit to the relative sender and receiver speeds.

The *maximum send rate* (respectively, *maximum receive*) in a system, denoted λ (respectively, μ) is the maximum rate at which a sending host can introduce datagrams into (remove datagrams from) a transmission medium, ignoring undelivered and corrupted datagrams. Let $\rho = \lambda/\mu$ denote the ratio of send rate to receive rate.

Application level TCP/IP throughput is limited by one of four factors: transmission medium data rate, λ, μ, and the TCP receive buffer size. We will assume that the first factor is infinite, which is reasonable for future high data rate networking technologies that are not a bottleneck.

The objective in this section is to measure how far away from the optimal data rate that TCP/IP functions for many of the hosts in Table 1. This can be represented by graphs of throughput as a function of ρ (Figures 8 and 9).

Experimental environment: The MacII, D310, PS/2, DEC2100, and DEC5000 are used; these provide several data points in speed from the slowest to the fastest hosts from Table 1. All

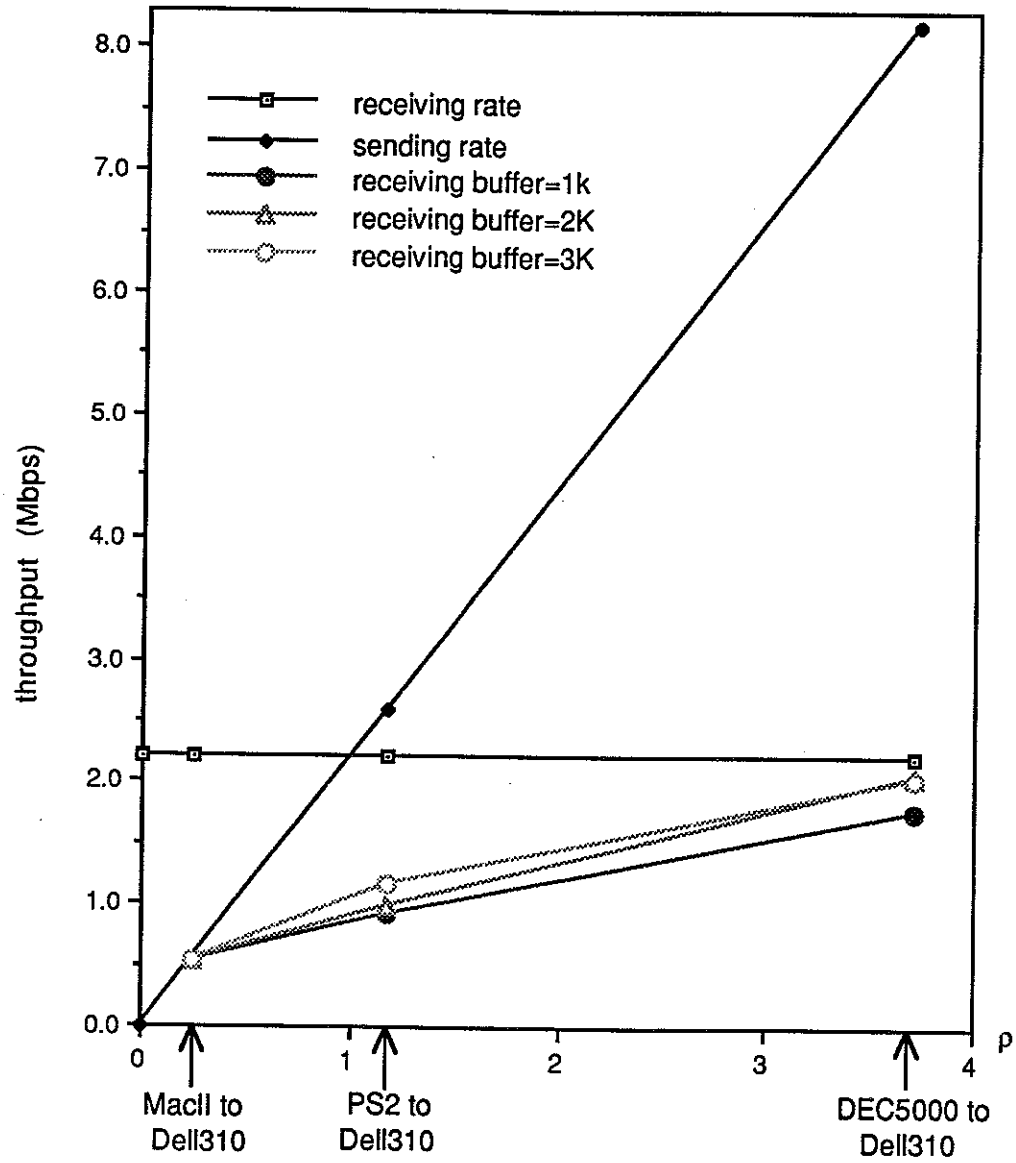

Figure 8: Optimal and measured TCP throughput for various speed hosts sending to the D310.

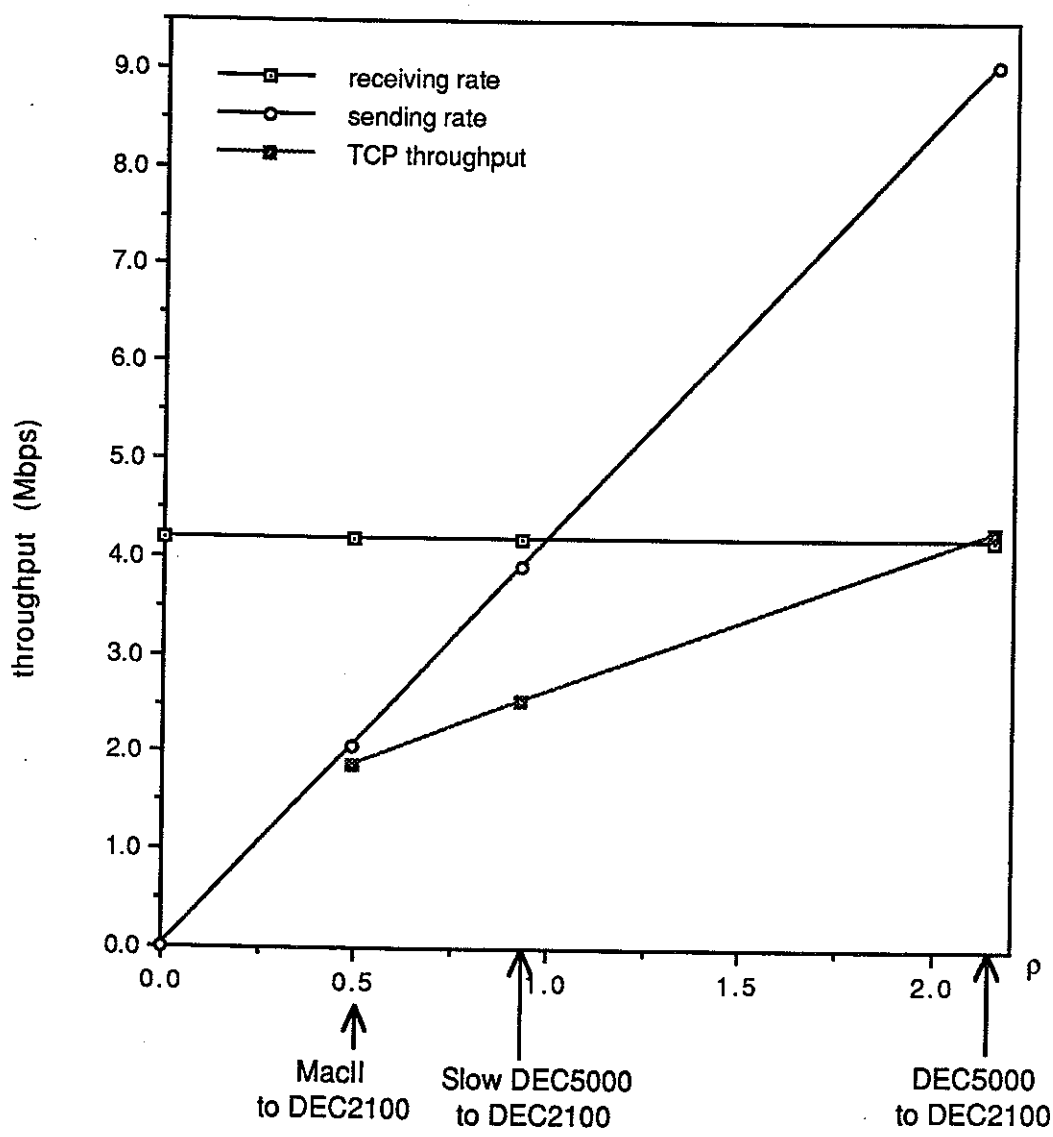

Figure 9: Optimal and measured TCP throughput for various speed hosts sending to the DEC2100.

experiments used a write size of 1K.

The maximum send rate of a host is estimated to be the throughput obtained using the following experiment. The sender runs the 40K byte file transfer program of Section 2.1 with UDP as the transport protocol and the DEC5000 as the receiver. The DEC5000 is faster than all other hosts used in this section, according to Figure 2. Therefore the DEC5000 can be viewed as an infinitely fast receiver. The measured throughput is then used as the maximum send rate. UDP is used as the transport protocol because there is no window mechanism to artificially limit throughput.

Only two hosts, the slower D310 and faster DEC2100, are used as receivers in the experiments. The D310 is used because its TCP receive window size, which is identified as a bottleneck in Section 3.3, can be changed, although over a limited range (1 to 3 K bytes).

The maximum receive of the D310 is estimated using measurements. The experiment used to compile Table 6 is used to measure the entire time taken to process a 1024 byte data segment, excluding only the application time. Therefore the time includes the processing time of TCP, IP, and the network driver. The receiver requires 3.469 milliseconds to process one data segment, and 0.477 milliseconds to generate and copy to the network adapter an acknowledgment segment. Assuming that every other data segment is acknowledged, the time required to process a 1024 byte packet is at least 3.71 milliseconds ($3.469 + 0.5 \times 0.477$), which corresponds to a maximum throughput of 2.209 Mbps.

The maximum receive rate for the DEC2100 is estimated by measuring the rate at which the host can accept UDP datagrams with 1K byte segments from a faster host, the DEC5000, without losing datagrams. The rate is 4.2 Mbps.

The aforementioned graphs of throughput as a function of ρ are constructed by collecting observations of throughput from 100 repetitions of the 40K byte file transfer application. The

sample mean of observations are reported.

Comparing measured, optimal throughput: Let $f(\lambda, \mu)$ denote the throughput obtained with send rate λ and receive μ. An upper bound on throughput is always given by the function $f(\lambda, \mu) = \min(\lambda, \mu)$. That is, if the sender is slower, then λ limits throughput, otherwise μ limits throughput.

Figure 8 presents the graph for a slow receiver, namely the D310 ($\mu = 2.209$ Mbps). The diagonal line is $f(\lambda, \mu) = \lambda$, and the horizontal line is $f(\lambda, \mu) = \mu$. The remaining curves connect points representing the throughput measured for senders that are half as fast ($\rho = 0.5$), equal to ($\rho = 1$), and slightly under four times as fast ($\rho = 3.7$) as the receiver for receive buffer sizes of 1K, 2K, and 3K.

Because the D310 is the slowest machine in Table 1, we artificially slow down the MacII to a send rate smaller than the D310 receive rate by inserting some extra operations between each *write*. The maximum send rates of the slowed down MacII, unaltered PS2, and unaltered DEC5000 are 0.552, 2.596, and 8.192 Mbps, respectively.

The distance from the three curves of actual throughput from the two upper bound curves suggests how far from optimal that TCP/IP performs. For the case of the MacII sending to the D310, Figure 8 shows that the throughput does not change with variations in receive buffer size. This is because the sender, not the receiver, is a bottleneck for small ρ. For the case of the PS2 sending to the D310, increasing the receive buffer size produces some improvement in throughput. However the connection operates far below the upper throughput bound. For the DEC5000 sending to the D310, increasing the buffer size from 1K to 2K increases throughput from 1.677 Mbps to 2.06 Mbps. The 1K case must have lower performance because each datagram contains 1K of data, and the protocol reduces to stop-and-wait. The 2K case permits a datagram

to be in transit while the receiving protocol stack processes a datagram arriving earlier. Because the receiver is several times slower than the sender ($\lambda = 8.192$ Mbps, $\mu = 2.209$ Mbps), increasing the receiver buffer size to 3K does not significantly improve throughput, because as long as the receiver can always find one packet in its receive buffer, the receive buffer size will not limit throughput.

The D310 is a slow machine by today's standards. Therefore we compare measured and optimal throughput for an early RISC workstation, the DEC2100 (for which $\mu = 4.2$ Mbps) in Figure 9. The sending machines are an unaltered MacII ($\lambda = 2.074$), a DEC5000 slowed down to $\lambda = 3.9$ Mbps by inserting extra operations between each *write*, and an unaltered DEC5000 ($\lambda = 9.04$ Mbps).

For the unaltered MacII sending to the DEC2100 ($\rho = 2.03$), Figure 9 shows that the measured throughput is nearly equal to the optimal throughput. For the slow DEC5000 sending to the DEC2100 ($\rho = 1.07$), the measured throughput is about half of the plotted upper bound on throughput. The TCP receive buffer is now the bottleneck. For the unaltered DEC5000 sending to the slower DEC2100 ($\rho = 2.15$), throughput reaches the optimum, because the TCP receive buffer is large enough to guarantee that the faster sender has deposited at least one datagram between the times at which the receiver removes datagrams.

4 Conclusions

The primary factor limiting performance for bulk data transfer applications over LANs in TCP/IP is the window flow control mechanism coupled with the strategy of setting the receiver advised window size to the currently available TCP receive buffer space. In fact, the more closely matched in data rates that a sender and receiver are, the larger the performance degradation becomes. Only after this bottleneck do the issues of time spent in memory copies

and checksumming and the operating system and network adapter interfaces become important, as reported by Cabrera *et al.* [2] and Clark *et al.*[5] Our measurements confirm what Clark *et al.* contend from statement counting: that the actual costs of executing the TCP and IP protocols themselves account for a very small amount of data transfer overhead.

Our measurements suggest that a file transfer between two hosts rated at x MIPs running TCP/IP but with an optimal flow control mechanism could at best fully utilize a quiescent network link with bandwidth $10x$ Mbps. This is encouraging, because when the current evolutions of RISC computer and gigabit networking architectures reaches their limit, we may be left with, in orders of magnitude, a 100 MIP workstation attached to and fully utilizing a gigabit network link.

What is the "optimal flow control mechanism"? Our measurements reveal the following. If the send and receive data rates are mismatched, TCP will probably operate near the optimum. But when the rates are closely matched, TCP will operate far below optimum unless the TCP receive buffer is properly sized.

The bottleneck analysis graphs (Figures 8, 9) show that with mismatched speed senders and receivers, TCP/IP performs close to optimal. The degree of mismatch which is needed before TCP/IP performs optimally varies. For the DECstation 21000 as receiver, the sender should at least twice as fast or at most half as fast as the receiver. For the 12.5 MHz Intel 80386-based DELL 310 PC as receiver, the sender should be at least four times as fast or at most one quarter as fast as the receiver.

But when the send and receive rates are close, TCP will artificially limit throughput unless the TCP receive buffer is carefully sized. Figures 8 and 9 show that throughput is cut in half in the DECstation and DELL, due to inadequate TCP receive buffer sizing. The DELL 310 is idle 45.3% of the time during a file transfer from an 80386-based IBM PS/2. The halving of

throughput and high idle time arise only because of window protocol coupled with inadequate TCP receive buffer sizing. An alternate flow control mechanism could do better.

An optimal flow control policy would "track" the upper bound curves in Figures 8 and 9. Considering the case of a file transfer between two hosts that do not communicate with other hosts during the transfer, rate based flow control is ideal. In this case, one can measure the maximum send and receive rates, λ and μ, of a host before attaching it to a network, just as we did in our study. The value of λ and μ only change when the host itself is altered, for example by increasing its memory or upgrading its processor or network adapter. A rate based sender can ask a receiving host for its value of μ at connection establishment time, and then transmit a rate that lies on the upper bound curve in Figures 8 and 9, namely $\max(\lambda, \mu)$, to achieve optimal throughput.

In contrast TCP cannot operate on the optimal curve because the protocol provides no way of learning or dynamically measuring λ and μ, although it might be possible to modify the protocol to do so. To use the information, however, TCP implementations will need a sophisticated algorithm to return receiver advised window sizes that could be larger than the amount of available receive buffer space, but will limit the likelihood of buffer overflow.

Our study also suggests that memory could be used more optimally in a TCP implementation which can learn about the relative speeds of senders and receivers, as rate-based flow control requires. For $\rho << 1$ and $\rho >> 1$, for the experiments reported here, a receiver buffer equal to just twice the maximum segment size is adequate to achieve optimal throughput.

Of course the fact that a transport protocol must adapt to varying traffic as connections on a host are established and released, complicates the problem of designing a flow control mechanism that tracks the optimal throughput curve. Further complication will arise in optimizing performance for interactive traffic, in contrast to the file transfer application studied here. Our future

research will investigate these problems in the framework of a large TCP window augmented within which a rate based control operates.

Acknowledgements

This research was supported by Research Initiation and Creative Match Grants from Virginia Polytechnic Institute and State University, an equipment donation from the IBM Zurich Research Laboratory, and equipment made available by E. Fox and Radford University. The assistance of L. Svobodova in making available the hardware clock used for measurements greatly facilitated this study.

References

[1] M. Abrams, *Design of a Measurement Instrument for Distributed Systems,* Research Report RZ1639, IBM Research Division, Zurich Research Laboratory, 1987.

[2] L. S. Cabrera, E. Hunter, M. J. Karels and D. A. Mosher, "User-Process Communication Performance in Networks of Computers," *IEEE Trans. on Software Engineering,* Vol. SE-14, No. 1, Jan. 1988.

[3] D, D, Clark, *Window and Acknowledgement Strategy in TCP,* Internet Protocol Implementation Guide, Network Information Center, SRI International, Menlo Park, CA ARPA RFC-813, Aug. 1982.

[4] D. D. Clark, M. L. Lambert, L. Zhang, "NETBLT: A High Throughput Transport Protocol," *Proc. of the ACM-SIGCOMM'87,* ACM, Stowe, VT, Aug. 1987, pp. 353-359.

[5] D. D. Clark, V. Jacobson, J. Romkey, H. Salwen, "An Analysis of TCP Processing Overhead," *IEEE Communications Magazine,* June, 1989, pp. 23-29.

[6] V. Jacobson, "Congestion Avoidance and Control," *Proc. of the ACM SIGCOMM '88,* ACM, Stanford, CA, Aug. 1988, pp. 314-329.

[7] R. Jain, *Myths About Congestion Management in High-Speed Networks,* DEC-TR-726, Oct. 1990.

[8] K. A. Lantz, W. I. Nowicki, M. M. Theimer, "An Empirical Study of distributed Application Performance," *IEEE Trans. on Software Engineering,* Vol. SE-11, No. 10, Oct. 1985, pp. 1162-1173.

[9] J. Postel, *DOD Standard Transmission Control Protocol,* ARPA RFC-793, Sep. 1981.

[10] J. H. Saltzer, D. D. Clark, J. L. Romkey, W. C. Gramlich, "The Desktop Computer as a Network Participant," *IEEE Journal on Selected Areas in Communications*, SAC-3, May, 1985, pp. 468-477.

[11] D. Sanghi, *et al.*, *Performance Instrumentation of TCP*, UMIACS-TR-88-24, CS-TR-2009, April, 1988.

[12] D. Sanghi, *et al.*, *Instrumenting a TCP Implementation*, UMIACS-TR-88-50 and CS-TR-2061, July, 1988.

[13] D. Sanghi, *et al.*, *A TCP Instrumentation and Its Use in Evaluating Round-trip-time Estimators*, UMIACS-TR-90-38 and CS-TR-2432, March, 1990.

[14] L. Svobodova, "Measured Performance of Transport Service in LANs," *Computer Networks and ISDN Systems*, Vol. 18, No. 1, 1989, pp. 31-45.

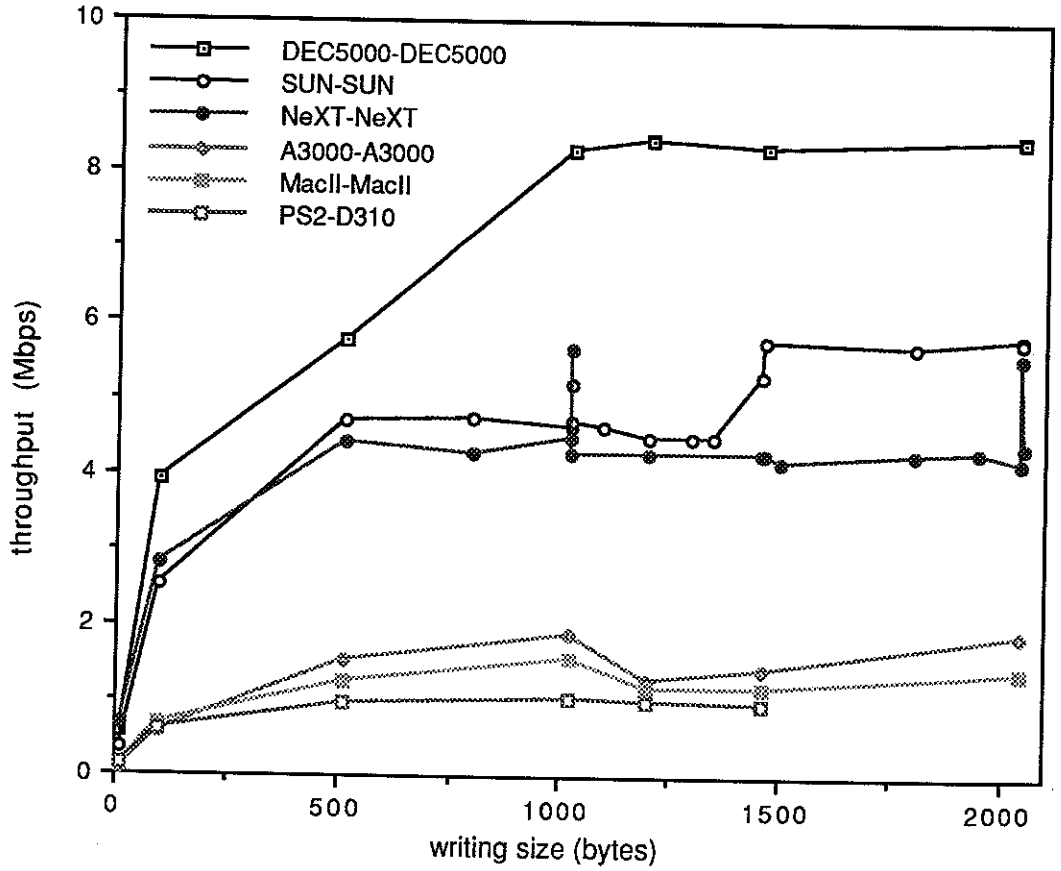

Figure 1: File transfer throughput using socket interface and TCP.

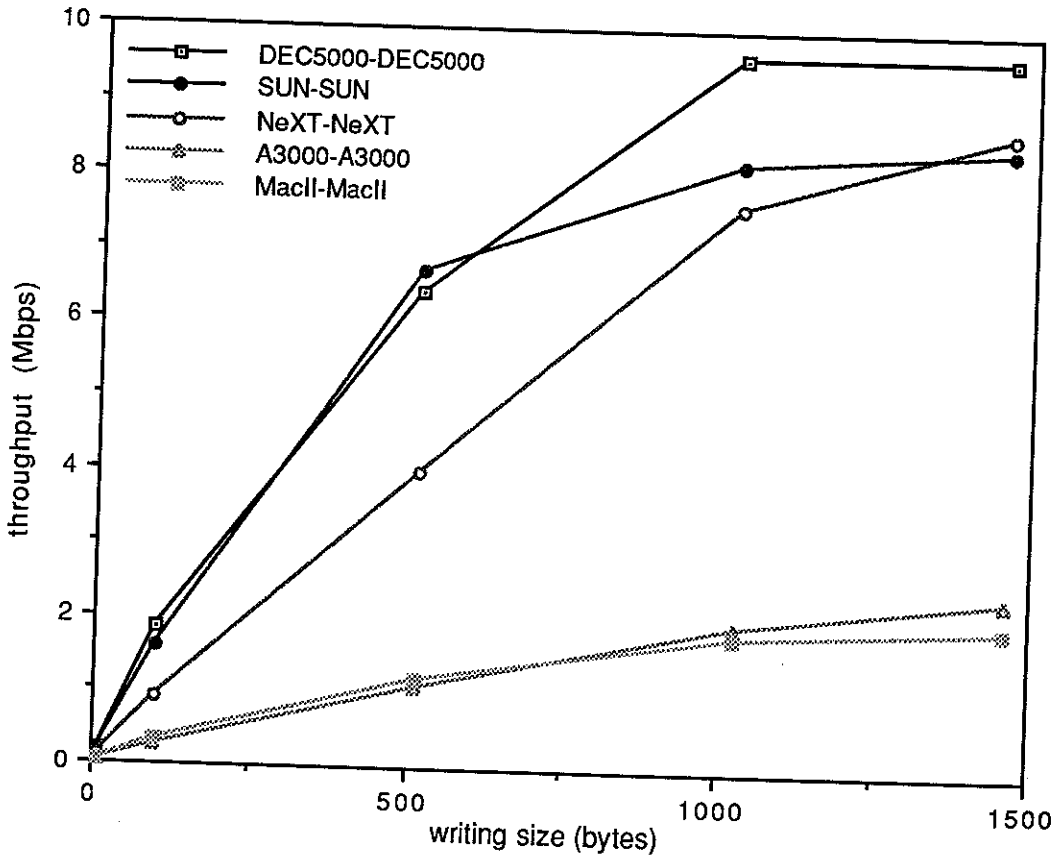

Figure 2: File transfer throughput using UDP and socket interface.

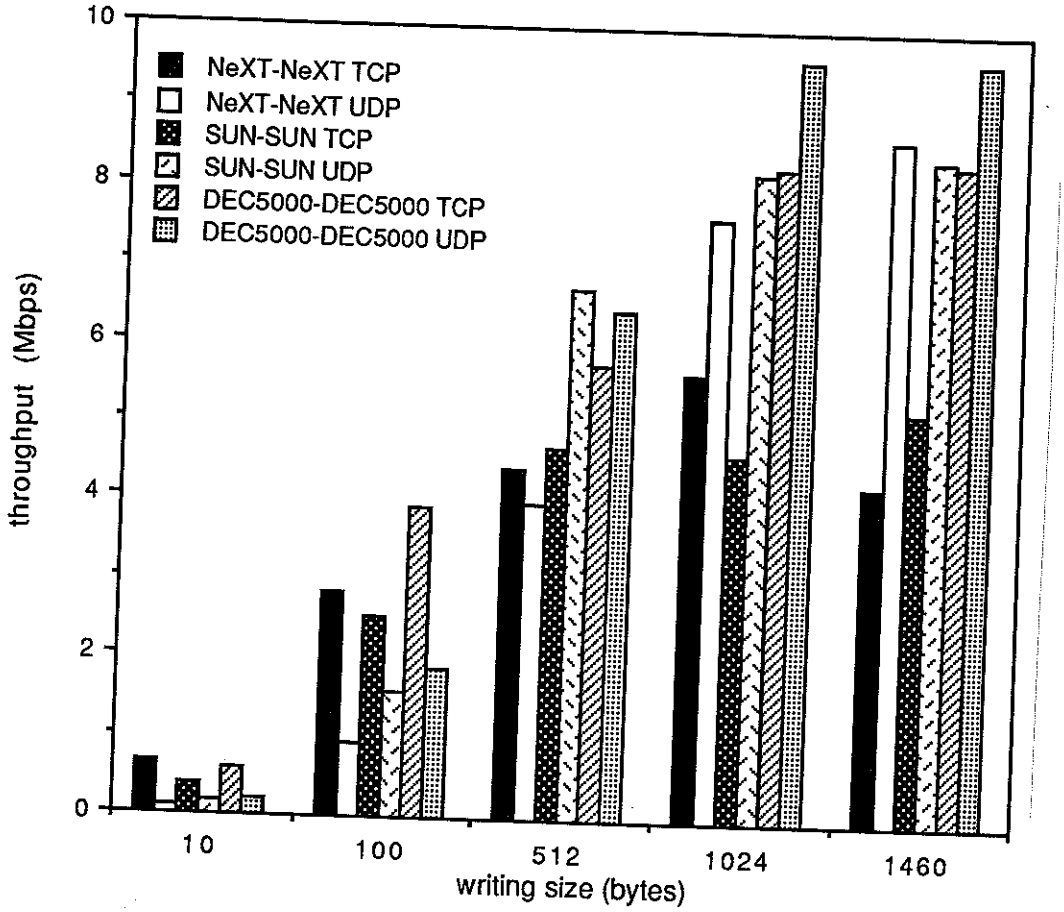

Figure 3: Comparison of TCP and UDP file transfer throughput using socket interface.

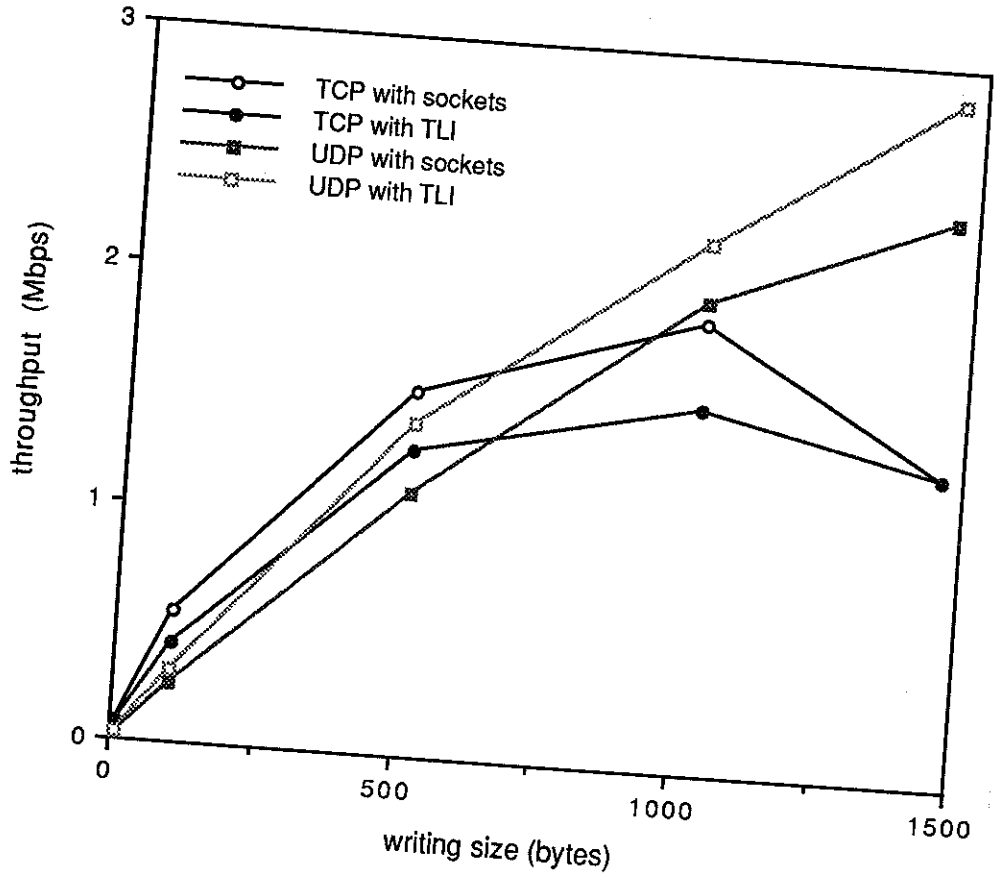

Figure 4: Comparison of socket and TLI interfaces on A3000 hosts.

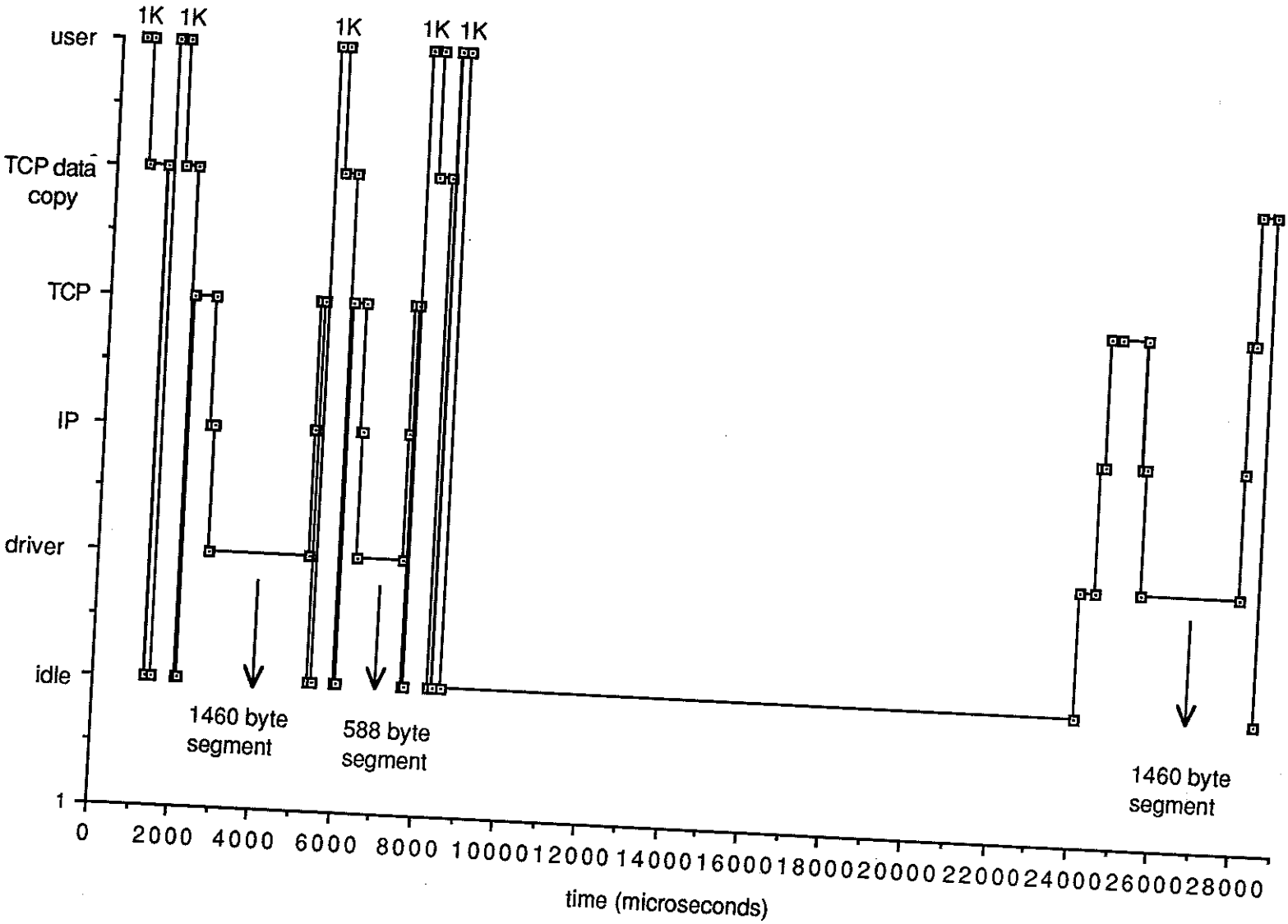

Figure 5: Time evolution of TCP/IP on the sending side for a transfer of 5K of data.

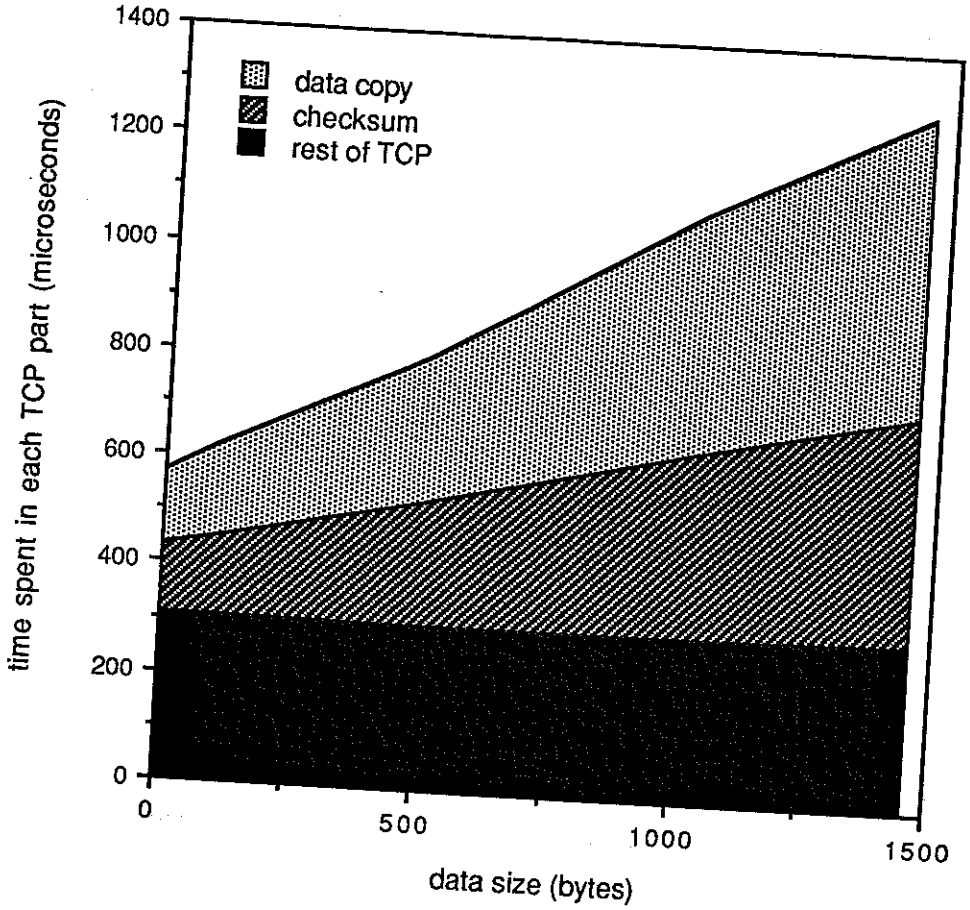

Figure 7: Overhead of TCP segment processing on the receiving side.

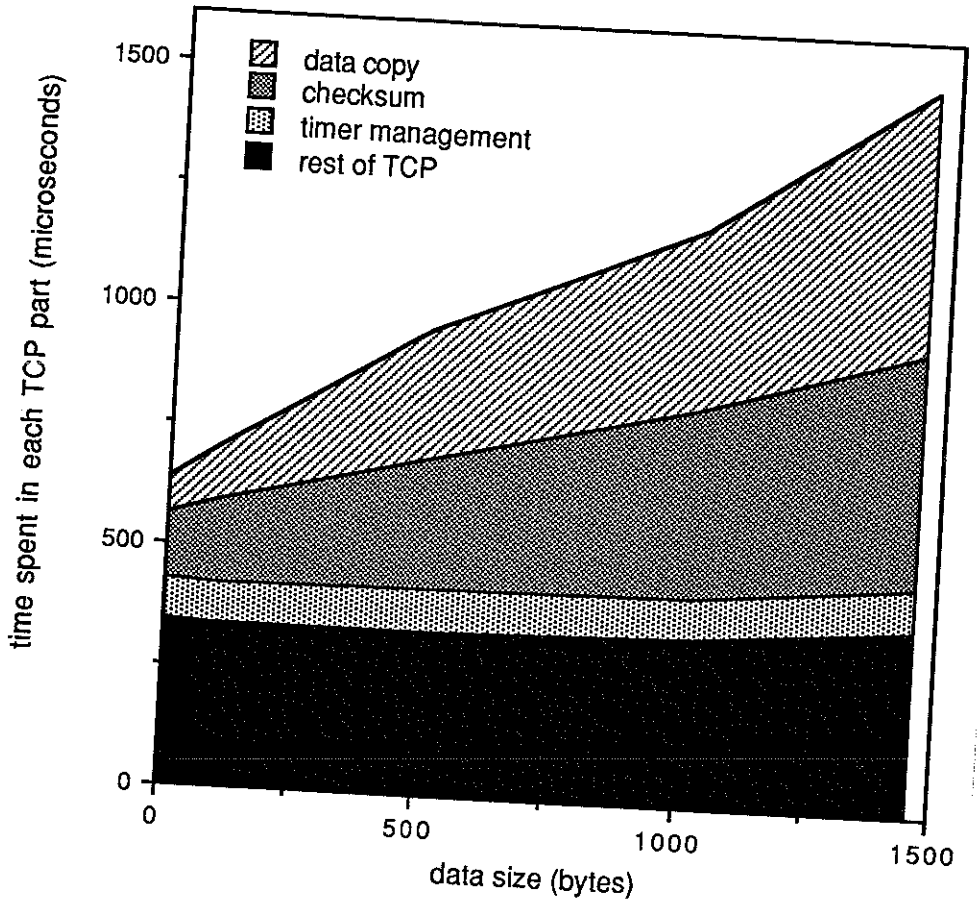

Figure 6: Overhead of TCP segment processing on the sending side.

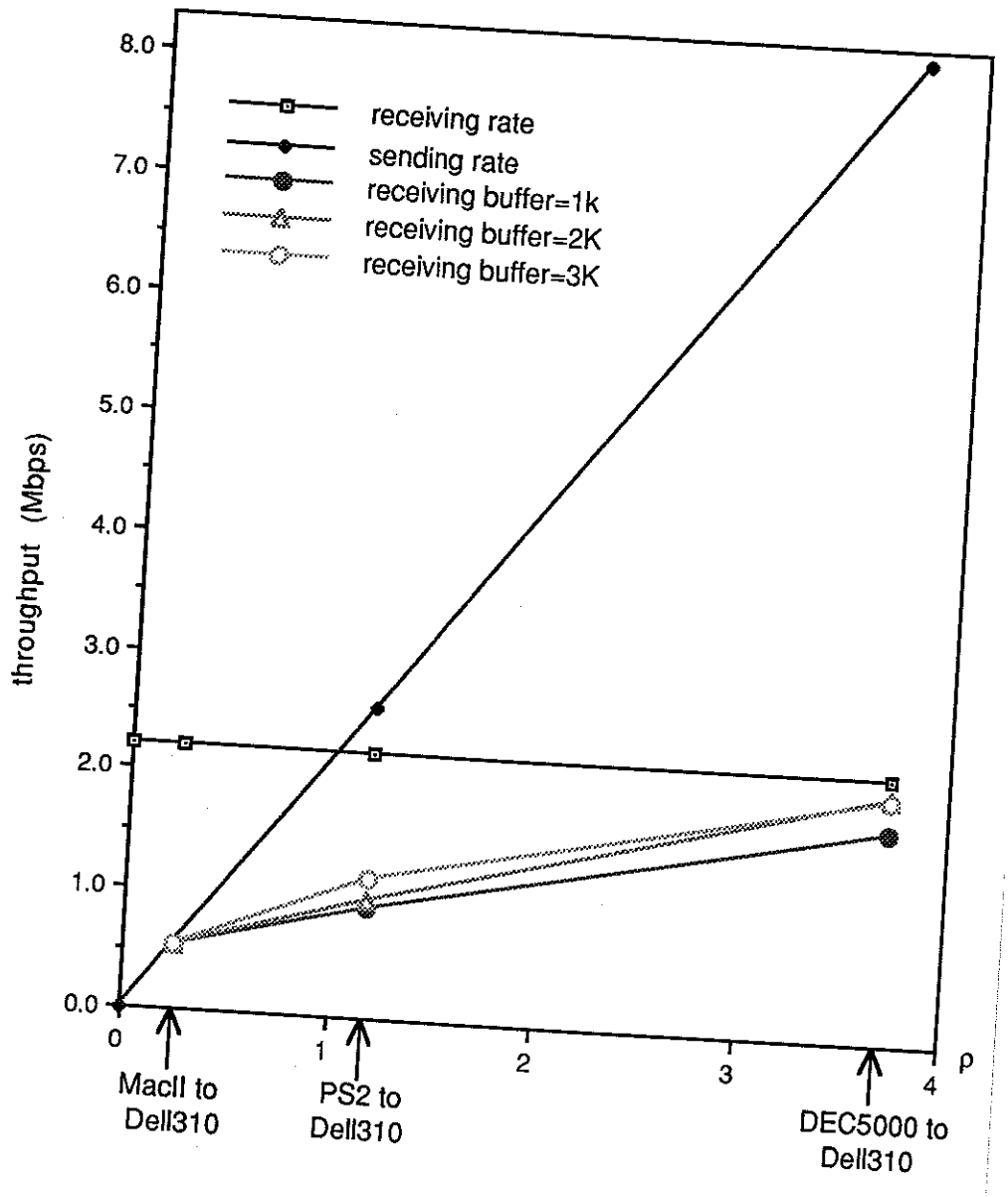

Figure 8: Optimal and measured TCP throughput for various speed hosts sending to the D310.

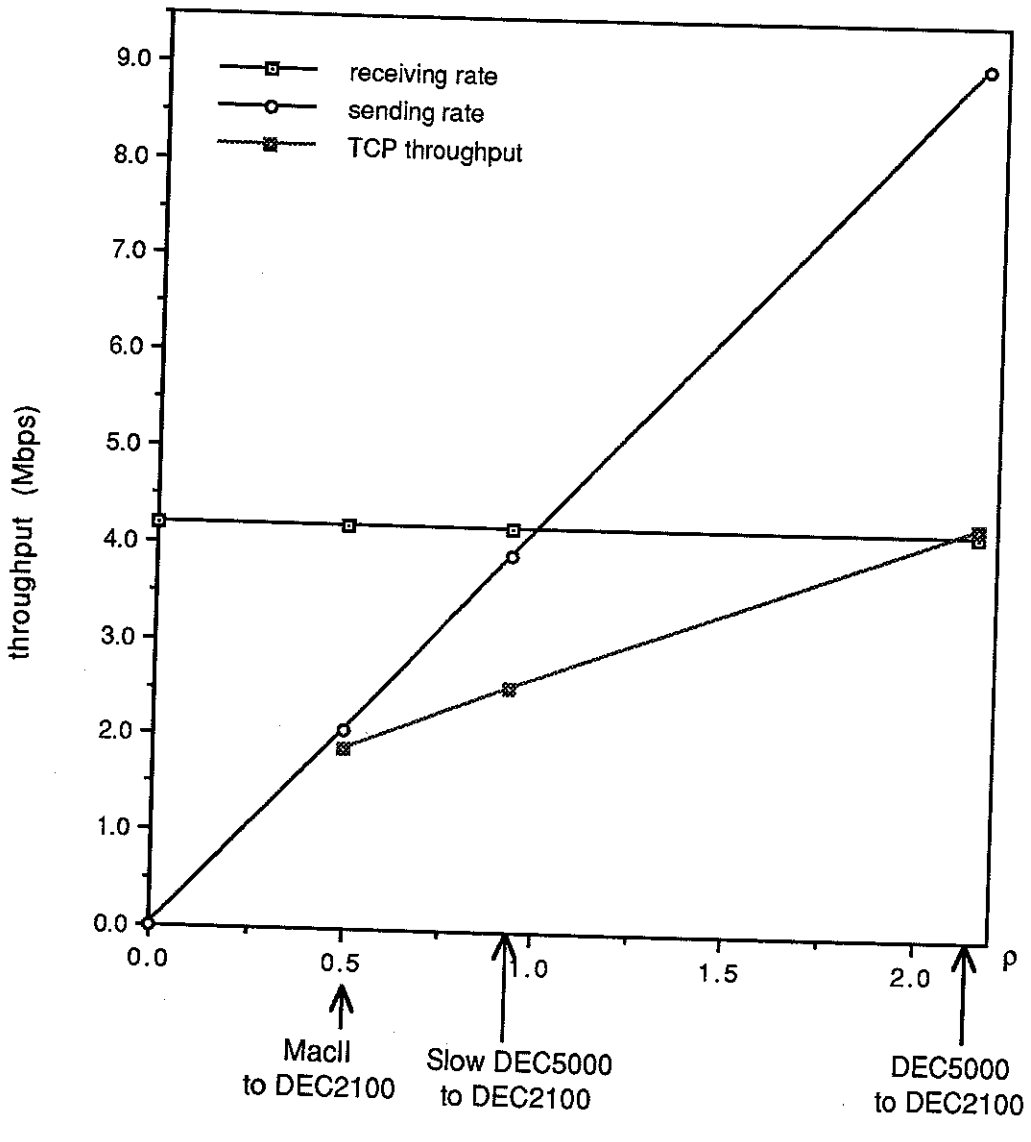

Figure 9: Optimal and measured TCP throughput for various speed hosts sending to the DEC2100.